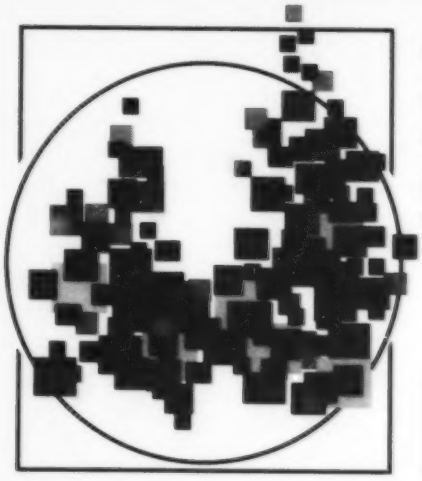


CENTRE FOR
SYSTEMS
SCIENCE

LCCR

Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 / (604) 291-4704

Distributed Database Concurrency Control by Cautious Schedulers

T. Ibaraki, S. Harashima,
M. Takeda, and T. Kameda

CSS/LCCR TR 89-04

92-05564/

Distributed Database Concurrency Control by Cautious Schedulers*

T. IBARAKI[1], S. HARASHIMA[2], M. TAKEDA[3] and T. KAMEDA[4]

CSS/LCCR TR 89-04

Abstract

Cautious schedulers have a very desirable feature that they prevent deadlock and rollbacks. This paper presents three concurrency control algorithms for distributed database systems based on cautious scheduling. The first algorithm is the most general and is meant for replicated databases, while the remaining two are for non-replicated databases. Each algorithm gives a higher priority to local transactions which need to access only local copies. They will be especially suitable for the case where there is a great deal of conflict among transactions and most of transactions are local. After some preparations, the three algorithms are presented and their correctness is proved.

* This work was supported in part by the Ministry of Education, Science and Culture of Japan under Scientific Grant-in-Aid, in part by the Advanced Systems Institute of British Columbia, and in part by the Natural Sciences and Engineering Research Council of Canada under Grant No. A5240.

(1) Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto, 606 Japan. (2) Systems and Software Engineering Laboratory, Toshiba, Ltd. Kawasaki, 210 Japan. (3) Ohita Office, Showa Denko, Ltd. Ohita, 870-01 Japan. (4) School of Computing Science, Simon Fraser University, British Columbia, Canada V5A 1S6. Netaddress: tiko%cs.sfu.ca@relay.ubc.ca.

1. Introduction

This paper treats distributed replicated database systems, although it also considers non-replicated database systems as a special case. Thus, we must deal with **concurrency control**, which guarantees serializability [BHG87, Pap86],[1] **replica control**, which hides replication, presenting one-copy view to user transactions, as well as **commit protocol** [BHG87] and **termination protocol** [ChK85, ChR86]. Although all these aspects must be integrated to design a successful distributed database system, our main interest in this paper is concurrency control. We will also briefly touch on replica control. Most of the concurrency control algorithms currently known for distributed database systems are based on two-phase locking or timestamping. Unfortunately, however, with many of these algorithms the system has to cope with occasional deadlocks and rollbacks, which is likely to degrade their performance considerably.

We propose a distributed concurrency control scheme based on **cautious scheduling** [CaB80, IKK88, IKK89], which is deadlock-free and does not require rollbacks, unless some transactions voluntarily abort themselves or a system failure forces some transactions to abort. Of course, the prevention of deadlocks and rollbacks incurs overhead, but we believe that in many applications where conflicts among transactions are frequent, cautious schedulers are viable contenders. Some quantitative studies, e.g., [NTI89], have been made to compare the throughput of different concurrency control schemes under realistic conditions.

In theory, replica control and concurrency control can be treated separately and each can be proved correct independently of the other, so that any correct concurrency control protocol can be combined with any correct replica control protocol [BHG87, GoL87]. However, there is a close interaction between the two, so that in practice they need to be considered together, especially if efficient implementation is desired. We adopt any of the replica control protocols based on the notion of **virtual partition** [EIT86, EIT89], which are resilient to partition failures, since they are best suited to our proposed concurrency control algorithms. One of our concurrency control algorithms, Algorithm A, incorporates some of replica control functions. In fact, within each "virtual partition", it decides which copies should be read or written. This intrusion into the "jurisdiction" of replica control is necessary, since we want to give preferential treatment to some transactions, depending on where the copies they access are located.

This paper proposes two algorithms, named A and B, and a modification of B, named C. Algorithm A is general in the sense that no restriction is imposed on transaction behavior. Each transaction that enters the system goes through two phases: the **registration phase** at its **home site** (where it is

[1] In this paper, "serializable" means "view-serializable" [Pap86]. Note that the "view" in this context has nothing to with the "view" of a site to be defined later.

submitted) and at other sites where the copies of data items it accesses are located, and the execution phase in which the transaction requests the execution of its steps. Algorithm B is more restricted: it is for non-replicated database systems in which each transaction is allowed to update only data items stored at its execution site, though it can read any data item. This algorithm is much simpler than Algorithm A, since its registration phase is greatly simplified. More specifically, all sites can operate independently of each other without requiring any message exchanges for coordination (coordination is automatically achieved by timestamps). In spirit, this is similar to the *Bakunin model* proposed in [KoG87]. Algorithm C is a modification of Algorithm B, in which "local transactions" accessing only the data items at one site enjoy a higher priority than global transactions. Since Algorithms B and C are for non-replicated databases, no replica control is required with them. (A commit protocol and termination protocol are still required to deal with failures.)

The next section gives an overview of the distributed database system model used in this paper. More details on cautious schedulers then follow in Sections 3 to 5. Algorithms A, B and C will then be discussed in Sections 6 to 11.

2. The Model

Our model closely resembles that used in ([BHG87] Chap. 8) with minor differences. Consider a distributed database system consisting of a number of sites. Let $D = \{ X, Y, ..., Z \}$ be the set of all **logical data items** which transactions access. Copies of each data item X are stored at some (one or more) sites. We use a subscripted lower case letter x_s to denote the copy of X stored at site s. Let D_s denote the set of all logical data items whose copies are stored at site s. The sets of copies corresponding to D and D_s are denoted $\overline{D}$ and $\overline{D}_s$, respectively.

Let $\mathbf{T} = \{T_0, T_1, T_2, ..., T_n\}$ be the set of all transactions to be executed in the system. T_0 and T_n are fictitious transactions called the **initial** and the **final transactions**. T_0 "writes" the initial values of all data items, while T_n "reads" their final values after all other transactions have completed. Each user **transaction** T_i $(1 \le i \le n-1)$ enters the system at some site in the system, which we call its **home site**, and issues requests for executing a sequence of **read steps** $R_i[S]$, $S \subseteq D$, and **write steps** $W_i[S']$, $S' \subseteq D$, which may be interleaved. A read step $R_i[S]$ stands for the set of read operations $\{R_i[X] \mid X \in S\}$; similarly for a write step. The operations in a step can be executed concurrently, possibly at different sites, but a transaction T_i is allowed to issue its next step only after the current step is completed. More formally, transaction T_i is a totally ordered set $(T_i, <_i)$, where $<_i$ is a total order on T_i,[2] such that if

[2] Here, T_i is considered as a *set* of logical steps as in [BHG87], as well as the *name* of a transaction.

T_i contains $R_i[S]$ and $W_i[S']$ with $S \cap S' \neq \emptyset$, then $R_i[S] <_i W_i[S']$. Let **writeset** $WS(T_i)$ (resp. **readset** $RS(T_i)$) be the set of logical data items which T_i writes (resp. reads). Cautious schedulers require that each transaction predeclare its readset and writeset before issuing its requests [CaB80, IKK88]. In the rest of the paper, for the sake of simplicity, we assume that each step accesses just one data item unless otherwise stated.

Each logical read operation $R_i[X]$ is translated by a replica control protocol into a **physical read operation** $r_i[x_s]$ on a copy, x_s, of X and each logical write operation $W_i[X]$ is translated into one or more **physical write operations** $\{w_i[x_s]\}$ on copies of X. More formally, the execution of a set of transactions in a replicated distributed database system is modeled by a **replicated data (RD) log** (or **history**) [BHG87]. An RD log over a set $\mathbf{T} = \{(T_i, <_i) \mid i = 0, 1, ..., n\}$ of transactions is a partially ordered set $(L(\mathbf{T}), <)$ of physical operations such that

(1) $L(\mathbf{T}) = \bigcup_{i=0}^{n} \tau(T_i)$ for some translation function τ, which maps each logical read operation $R_i[X]$ into one physical read operation $r_i[x_s]$, and each logical write operation $W_i[X]$ into physical write operations on some copies of X;

(2) For each i and any two logical operations p_i and q_i in T_i, if $a \in \tau(p_i)$, $b \in \tau(q_i)$ and $p_i <_i q_i$, then $a < b$ (read "a **precedes** b");

(3) All pairs of conflicting physical operations are $<$ related, where two physical operations **conflict** if they operate on the same physical copy of a data item and at least one of them is a write operation; and

(4) The initial transaction T_0 is translated by τ into a set of physical write operations, one for each copy of each data item, and it precedes all other operations on that copy with respect to $<$. Similarly, the final transaction T_n is translated into a set of physical read operations, one for each logical data item. Such a read operation is preceded by all physical write operations on the copies of the corresponding logical data item.

All distributed systems are subject to communication failures, which may cause **partition failures**. Several replica control protocols resilient to partition failures have been proposed in the past [BHG87]. We adopt any of the **virtual partition protocols** originally proposed in [ESC85] and later expanded in [EIT86], [EIT89] and [FuK89]In these protocols, each site s maintains a set of sites called its **view**. Intuitively, the view of a site s contains all sites with which s assumes it can communicate, i.e., the partition that s thinks it belongs to, although it need not always reflect the reality. For this reason, a view is also called a **virtual partition**.

In quorum-based protocols [ESC85, Gif79, Her87], not all copies are in general read or written. A **(view) read quorum** for a data item X is a set of physical copies of X stored at sites in a view that are *accessed* (the value of one of them is actually read) to execute a logical read operation on X, and a (view) **write quorum** for a data item X is a set of physical copies of a X stored at sites in a view that are written to execute a logical write operation on X. In this paper, we adopt the "read-one-write-all" approach, i.e., a read quorum contains only one copy and a write quorum contains all copies within the current view. This is necessary to guarantee that the following algorithm A outputs only serializable schedules. In practice, read operations are likely to be more frequent than write operations, in which case this approach would lead to an efficient implementation.

Each transaction in an RD log can be viewed as a partially ordered set of physical operations. (Physical write operations corresponding to the same logical write operation may not be ordered by this partial order.) Let σ be a sequence of logical steps of all transactions in **T**, in which $W_0[D]$ is the first step, $R_n[D]$ is the last step, and all steps of each transaction T_i appear in the order issued by T_i. Let α be an RD log over **T**. The **version selection** or **interpretation** I (resp. $\overline{I}$) for σ (resp. α) is a function from the set of all logical read operations in σ (resp., physical read operations in α) into the set of logical write operations in σ (resp. physical write operations in α). Intuitively, if $I(R_i[X]) = W_k[X]$ (resp. $\overline{I}(r_i[x_a]) = w_k[x_a]$), this means that $R_i[X]$ (resp. $r_i[x_a]$) reads the value written by $W_k[X]$ (resp. $w_k[x_a]$). A pair $<\sigma, I>$ (resp. $<\alpha, \overline{I}>$) is called a **schedule** (resp. a **physical schedule**) if, for each $R_i[X]$ in σ (resp. $r_i[x_a]$ in α), $I(R_i[X])$ precedes $R_i[X]$ in σ (resp. $\overline{I}(r_i[x_a]) < r_i[x_a]$ in α). The **standard interpretation**, denoted I^*, for a schedule σ always maps a logical read operation $R_i[X]$ into the last logical write operation on X preceding $R_i[X]$ in σ. Similarly, the standard interpretation $\overline{I}^*$ for a physical schedule α always maps a physical read operation $r_i[x_a]$ into the last physical write operation on x_a preceding $r_i[x_a]$ in α. (By the defining condition (2) on an RD log, such a physical write operation is uniquely determined.)

Let $<\sigma, I>$ be a schedule and $<\alpha, \overline{I}>$ be a physical schedule. We say that $<\alpha, \overline{I}>$ **realizes** $<\sigma, I>$ if the following two conditions are met.

(1) For each read operation $R_i[X]$ in σ, there is one physical read operation $r_i[x_a]$, and for each write operation $W_i[X]$ in σ, there is at least one physical write operation $w_i[x_{a'}]$ in α.

(2) For each physical read operation $r_i[x_a]$ in α, if $\overline{I}(r_i[x_a]) = w_k[x_a]$ then $I(R_i[X]) = W_k[X]$.

Example 2.1 Consider a distributed database system on sites a and b, where $D = \{X, Y\}$ and $\overline{D} = \{x_a, y_a, y_b\}$. Let schedule $<\sigma, I>$ and physical schedule $<\alpha, \overline{I}>$ be defined as follows, where operations are listed from left to right in the order of their precedence. In this example, the projection of the partial

order < for α on each site happens to be a total order. Thus α is represented by two sequences, one for each site. The precedence between operations at different sites implied by the condition (2) on a RD log is not explicitly shown.

$$\sigma = W_0[D]\, R_1[X]\, R_2[Y]\, W_2[Y]\, R_3[Y]\, W_1[X]\, W_3[X]\, R_n[D],$$

$$I(R_1[X]) = W_0[X],\ I(R_2[Y]) = W_0[Y],\ I(R_3[Y]) = W_2[Y],\ I(R_n[X]) = W_3[X],\ I(R_n[Y]) = W_2[Y],$$

$$\alpha = \{\ w_0[\overline{D}_a]\, r_1[x_a]\, w_1[x_a]\, r_2[y_a]\, w_2[y_a]\, w_3[x_a]\, r_n[x_a, y_a];\ w_0[\overline{D}_b]\, w_2[y_b]\, r_3[y_b]\, r_n[\,]\ \}$$

$$\overline{I}(r_1[x_a]) = w_0[x_a],\ \overline{I}(r_2[y_a]) = w_0[y_a],\ \overline{I}(r_3[y_b]) = w_2[y_b],\ \overline{I}(r_n[x_a]) = w_3[x_a],\ \overline{I}(r_n[y_a]) = w_2[y_a].$$

Note that the defining conditions (1), (2) and (3) on an RD log hold for α. In particular, the final transaction reads one copy of each data item. It is easy to see that $<\alpha, \overline{I}>$ realizes $<\sigma, I>$. □

The concept of **one-copy serializability** (1SR,[3] for short) [BHG87] is widely used as a correctness criterion for physical schedules. It follows easily from definition that a physical schedule $<\alpha, \overline{I}>$ is 1SR if and only if it realizes a serializable schedule [BHG87].

Our purpose in this paper is to design concurrency control protocols that produce only 1SR physical schedules. [ESC85, EIT86, EIT89, FuK89]present partition-failure resilient replica control protocols based on virtual partitions, which guarantee global 1SR provided that they are used together with a concurrency control protocol that generates 1SR physical schedules in each view (or virtual partition). In view of this, we concentrate on concurrency control in each view. The initial transaction T_0 then models the **view-update transaction** [EIT86] for the current view V, and if V doesn't contain enough copies of a data item, no transaction that needs to access the data item can be executed in V. Similarly, the final transaction T_n models the view-update transactions of subsequent views, which copy values updated in V.

3. TIO Graph and Serializability

This section introduces a few concepts, based on which it can be determined whether a given physical schedule is 1SR.

For a schedule $<\sigma, I>$ over the set **T** of transactions, the **TIO graph (transaction input/output graph), denoted** $TIO(\sigma, I)$, is defined as follows [IKM87]. $TIO(\sigma, I)$ is a labeled directed graph which has a node for each transaction $T_i \in$ **T** and a **reads-from** arc $(T_k, T_i):X$ (X is its label) if $I(R_i[X]) = (W_k[X])$. Also, if T_i has a write operation $W_i[X]$ for which there is no $R_j[X]$ such that $I(R_j[X]) = W_i[X]$ holds (such $W_i[X]$ is called a **useless** write), then $TIO(\sigma, I)$ has a **dummy node** T_i' with **dummy arc**

(T_i, T_i'):X. (T_i is called the **parent** of T_i'.) $TIO(\sigma, I)$ has no other node or arc.

A total order $\ll$ defined over the set of nodes of $TIO(\sigma, I)$ is called a **DITS (disjoint interval topological sort)** if

(i) $T_i \ll T_j$ implies that there is no path in $TIO(\sigma, I)$ from T_j to T_i, and

(ii) *Exclusion rule*: For two arcs with the same label, (T_g, T_h):X and (T_i, T_j):X such that $T_g \neq T_i$, $T_g \ll T_j$ implies $T_h \ll T_i$ (i.e., these two arcs do not overlap when nodes are arranged in the order given by $\ll$).

Theorem 3.1 [IKM87] A schedule $<\sigma, I>$ is serializable if and only if $TIO(\sigma, I)$ has a DITS that orders T_0 first and T_n last. □

Similarly to $TIO(\sigma, I)$, we define the **tio graph**, $tio(\alpha, \overline{I})$, for a physical schedule $<\alpha, \overline{I}>$ with a modification that label x_s is attached to arc (T_k, T_i) if $\overline{I}(r_i[x_s]) = w_k[x_s]$ or if (T_k, T_i) is a dummy arc associated with a useless write operation $w_k[x_s]$. A DITS for $tio(\alpha, \overline{I})$ is also similarly defined by modifying condition (ii) as follows.

(ii') *Exclusion rule*: For two arcs (T_g, T_h):x_s and (T_i, T_j):$x_{s'}$ ($s = s'$ is possible), where $T_g \neq T_i$, $T_g \ll T_j$ implies $T_h \ll T_i$.

Example 3.1 Figure 1 shows $TIO(\sigma, I)$ and $tio(\alpha, \overline{I})$ for Example 2.1. It is immediate to see that the following total order

$$T_0, T_1, T_1', T_2, T_3, T_n$$

is a DITS for both $TIO(\sigma, I)$ and $tio(\alpha, \overline{I})$, where T_1' is the dummy node associated with T_1. □

Theorem 3.2 A physical schedule $<\alpha, \overline{I}>$ is 1SR if and only if $tio(\alpha, \overline{I})$ has a DITS that orders T_0 first and T_n last.

Proof. Necessity. Let $<\alpha, \overline{I}>$ be 1SR and let it realize a serializable schedule $<\sigma, I>$. By Theorem 3.1, $TIO(\sigma, I)$ has a DITS since $<\sigma, I>$ is serializable. As $<\alpha, \overline{I}>$ realizes $<\sigma, I>$, there is an onto mapping from the nondummy arc set of $tio(\alpha, \overline{I})$ to the nondummy arc set of $TIO(\sigma, I)$ such that each arc (T_k, T_i):x_s is mapped to (T_k, T_i):X. This mapping associates each nondummy arc (T_k, T_i):X in $TIO(\sigma, I)$ with exactly one nondummy arc (T_k, T_i):x_s by condition (1) for realization. As dummy nodes in $tio(\alpha, \overline{I})$ can be ordered (with respect to <<) immediately after their parent nodes without violating the DITS conditions (i) and (ii'), this shows that a DITS for $TIO(\sigma, I)$ is also a DITS for $tio(\alpha, \overline{I})$.

Sufficiency. $TIO(\sigma, I)$ can be constructed from $tio(\alpha, \overline{I})$ by the following procedure: For each node $T_i \in \mathbf{T}$ and each $X \in D$, examine all the arcs of the form (T_i, T):x_s for some site s. If all these are

dummy arcs, replace them with a single dummy arc (T_i, T_i'):X. Otherwise, replace all nondummy arcs $\{(T_i, T_j)$:$x_s \mid s$ is a site$\}$ with a single arc (T_i, T_j):X and ignore all dummy arcs $\{(T_i, T_i')$:$x_{s'} \mid s'$ is a site$\}$. In this case, if $tio(\alpha, \bar{I})$ has two arcs, (T_i, T_j):x_s and (T_k, T_j):$x_{s'}$, then $i = k$ since $tio(\alpha, \bar{I})$ has a DITS. Furthermore, the set $\{(T_i, T_j)$:$x_s \mid s$ is a site$\}$ contains exactly one arc by condition (1) for RD log α. Therefore, the above (T_i, T_j):X is well-defined. It is easy to see that the resulting graph is $TIO(\sigma, I)$ that is realized by $tio(\alpha, \bar{I})$. Therefore, any DITS for $tio(\alpha, \bar{I})$ is a DITS for $TIO(\sigma, I)$. □

4. Cautious Schedulers

We use a cautious scheduler at each site to process the incoming requests from transactions. We first review some basic properties of cautious schedulers [CaB80, IKK88, IKK89]. Assume for the time being that the system is centralized (i.e., there is just one site s). A cautious scheduler $CS(C)$ is defined for a class C of schedules, which is a subclass of the class of all serializable schedules. Each transaction T_i is required to predeclare $WS(T_i)$ and $RS(T_i)$ before its execution begins. $CS(C)$ arranges on-line all the incoming steps issued by transactions into a sequence α and determines interpretation $\bar{I}$, so that a physical schedule $<\alpha, \bar{I}>$ that is eventually output realizes a serializable schedule $<\sigma, I> \in C$.

A key step of cautious scheduler $CS(C)$ is the **completion test**: Given a partial schedule $<P, \bar{I}>$ already output by $CS(C)$, the current request q, and the set $PEND$ of **pending operations** (some operations in $PEND$ may have already arrived but are being delayed, while others are being anticipated according to $WS(T_i)$ and $RS(T_i)$ predeclared by T_i), $CS(C)$ tests whether $<P, \bar{I}>$ can be extended to $<PqQr_n[D], \bar{I}\ \bar{I}'>$ that realizes a schedule in C, where Q is a sequence consisting of all the operations in $PEND$ and $\bar{I}'$ is interpretation for the augmented sequence $qQr_n[D]$.

It is not always possible to perform efficient completion test for a given class C. In some cases, such a test is known to be NP-complete. Among the practically important classes, for which completion test can be done in polynomial time, are two classes WW and MWW (a multiversion class) [IKK88, IKK89]. The completion test for these classes is done by using the **active TIO graph (ATIO graph**, for short), $atio(<P, \bar{I}>, q, PEND)$, which is defined similarly to $tio(P, \bar{I})$ except that any $w_i[x_s] \in PEND \cup \{q\}$ is represented by a dummy arc (T_i, T_i'):x_s. Depending upon class C, some additional constraint arcs are introduced as follows.

(a) For $C = WW, MWW$, add a *ww*-constraint arc (T_i, T_j) if $w_i[x_s]$ precedes $w_j[x_s]$ in the sequence Pq, or if $w_i[x_s]$ belongs to Pq and $w_j[x_s]$ belongs to $PEND$.

---

[3] 1SR will also mean "one-copy serializable".

(b) For $C = WW$, add a wr-constraint arc (T_k, T_j) if $w_k[x_s]$ is the last write operation on x_s in Pq and $r_j[x_s] \in PEND$.

(c) For $C = WW$, add a reads-form arc $(T_k, T_i):x_s$ if $r_i[x_s] \in q$ and $w_k[x_s]$ is the last write operation on x_s in P.

We also use **timestamp constraint arcs**, which are not needed in the original cautious scheduling but are necessary to carry out distributed concurrency control. If T_i has timestamp (see Section 5) t_i and T_j has t_j, such that $t_i < t_j$, then we add a constraint arc (T_i, T_j). The **exclusion closure** of $atio(<P, \overline{I}>, q, PEND)$ is then obtained by adding the **exclusion arc** (T_k, T_i) for any pair of arcs $(T_g, T_k):x_s$ and $(T_i, T_j):x_s$ such that $T_g \neq T_i$ and there is a path from T_g to T_j. The resulting graph is denoted $atio^*(<P, \overline{I}>, q, PEND)$.

As seen from the above rules, $CS(WW)$ differs from **multiversion** cautious scheduler $MCS(MWW)$ in its version selection (or interpretation). $CS(WW)$ selects for each $r_i(x_s)$ the most recent version of copy x_s, while $MCS(MWW)$ is more flexible in selecting versions.

Theorem 4.1 [IKK88, IKK89] Given $<P, \overline{I}>$, q, $PEND$ as defined above, the completion test of $CS(WW)$ and $MCS(MWW)$ succeeds if and only if the corresponding $atio^*(<P, \overline{I}>, q, PEND)$ is acyclic. □

Therefore, the completion test can be done in polynomial time. If the completion test succeeds, $CS(C)$ or $MCS(MC)$ grants q for execution; but if it fails, q is placed in queue DEL, in which all the operations delayed so far are stored (DEL forms a subset of $PEND$). Whenever the completion test succeeds for some operation q, all operations in DEL are tested anew to see if they can now be granted. In this way, it can be shown that any operation in DEL will be eventually granted.

In summary, it is known that a cautious scheduler $CS(C)$ and $MCS(MC)$ have the following properties.

(1) No rollback is necessary for the purpose of concurrency control.

(2) Any physical operation sent to $CS(C)$ or $MCS(MC)$ is eventually executed in finite time (i.e., no deadlocks).

5. Assumptions on Distributed Database Management Systems

We describe here the distributed database management system, with which our concurrency control algorithms will be designed in the subsequent sections. In this and the next two sections, $CS_s(C)$ will stand for either a single-version or a multiversion cautious scheduler at site s.

(1) There is a number of sites in the system, every pair of which is connected by a communication link. Each site s has its identification number $id(s)$, and is equipped with a communication controller CC_s, a transaction manager TM_s, a cautious scheduler $CS_s(C)$, a data manager DM_s, and a set of copies $\overline{D}_s$, as illustrated in Fig. 2.

(2) Each site s has its own clock, and can issue timestamp $t = (cl_s, id(s))$, where cl_s is the current value of its clock. The second part $id(s)$ is added to make timestamp t unique in the entire system.

(3) Each site s knows the sets of copies $\overline{D}_l$ at all sites l.

6. Outline of Algorithm A

Each transaction T_i enters the system at one site, $home(T_i)$, which we call the **home site** of T_i. T_i first predeclares its $WS(T_i)$ and $RS(T_i)$, based on which the home site selects quorums as described in Section 2. Let $ws(T_i)$ and $rs(T_i)$ denote the unions of the selected write quorums and read quorums, respectively. If there is a site s such that $ws(T_i) \cup rs(T_i) \subseteq \overline{D}_s$,[4] T_i is called a **local transaction**, and s the **execution site** of T_i, denoted $ex_site(T_i)$; otherwise, T_i is a **global transaction**.

If T_i is a local transaction, it is directly registered with the cautious scheduler at $ex_site(T_i)$ in the same manner as in a centralized database system. No timestamps are necessary for local transactions. All steps issued by T_i thereafter are sent to $ex_site(T_i)$ and processed there as described in Section 4. It is possible to have $ex_site(T_i) = home(T_i)$. If this is the case for a local transaction T_i, then no communication is necessary between $ex_site(T_i)$ and other sites to execute T_i.

If T_i is a global transaction, on the other hand, $s = home(T_i)$ sends messages $(T_i, ws(T_i) \cap \overline{D}_l, rs(T_i) \cap \overline{D}_l)$ to all the **relevant** sites l, i.e., those sites satisfying $\overline{D}_l \cap (ws(T_i) \cup rs(T_i)) \neq \emptyset$. Upon receiving this message, transaction manager TM_l at site l registers T_i with $CS_l(C)$, i.e., adds node T_i to its ATIO graph, $atio_l^*(\cdot)$, with writeset $ws(T_i) \cap \overline{D}_l$, readset $rs(T_i) \cap \overline{D}_l$ and a tentative timestamp $t_l(Ti) = (cl_l, id(l))$ (as well as all the constraint arcs induced). It will be shown later (Section 8.1) that $atio_l^*(\cdot)$ in this case is always acyclic. Site l then sends back to $home(T_i)$ a message $(T_i, t_l(T_i))$. The process up to this point is depicted in Fig. 3 (a) and (b).

Upon receiving reply messages $(T_i, t_l(T_i))$ from all the relevant sites l, $home(T_i)$ determines

$$t_{max}(T_i) = \max_l t_l(T_i)$$

and uses it as the timestamp of T_i. This completes the registration phase of T_i. Thereafter, whenever T_i requests the execution of its next step, $home(T_i)$ sends the corresponding physical operations to the appropriate sites, where they are executed. If it is the first request to a site l, the timestamp $t_{max}(T_i)$

[4] If every data item is replicated, then all local transactions are read-only transactions.

accompanies the request; then $CS_l(T_i)$ adjusts the present (tentative) timestamp of T_i at site l to a permanent timestamp $t_{max}(T_i)$ (see Theorem 8.1). Each operation of T_i sent to site l will be eventually executed, and the result is sent back to $home(T_i)$. This is depicted in Fig. 3(c). As it was informed of $ws(T_i) \cap \overline{D}_l$ and $rs(T_i) \cap \overline{D}_l$ beforehand, site l can know when all physical operations of T_i at site l have been executed; transaction T_i is then treated as completed in $CS_l(C)$.

7. Description of Algorithm A

To describe Algorithm A more precisely, we itemize the steps that the transaction manager at site s, TM_s, carries out in response to various requests from other sites and from transactions.

A1. *Arrival of a transaction* T_i. Site s becomes $home(T_i)$, and carries out the following registration phase.

(1) Given write set $WS(T_i)$ and read set $RS(T_i)$ predeclared by T_i, it selects appropriate read and write quorums for all data items in $WS(T_i)$ and $RS(T_i)$, as described in Section 2, to obtain $ws(T_i)$ and $rs(T_i)$.

(2) If T_i is a local transaction with $l = ex_site(T_i)$, send message $(T_i, ws(T_i), rs(T_i))$ to l for registration.

(3) If T_i is a global transaction, send a message $(T_i, ws(T_i) \cap \overline{D}_l, rs(T_i) \cap \overline{D}_l)$ to each relevant site l for registration. Upon receiving reply messages $(T_i, t_l(T_i))$ from all relevant sites, define

$$t_{max}(T_i) = \max_l t_l(T_i)$$

as the timestamp of T_i. This completes the registration phase of T_i. At this point, if the current clock cl_s of site s is smaller than the clock part of $t_{max}(T_i)$, advance cl_s to that value.

A2. *Submission of a step from* T_i $(s = home(T_i))$. Assume that the step is $W_i[S]$. (The case for $R_i[S]$ is similar.) Convert each operation $W_i[X]$, $X \in S$, into the set of physical operations $w_i[x_l]$, on the basis of quorum selection in A1(1) (see Section 2).

(1) If T_i is a local transaction with $l = ex_site(T_i)$, send all physical operations $w_i[x_l]$ to site l for execution.

(2) If T_i is a global transaction, send the corresponding physical operations $w_i[x_l]$ to the sites l where their copies reside. If this is the first request of T_i to site l, send the timestamp $t_{max}(T_i)$ as well.

A3. *Registration of a local transaction* T_i *sent from site* l, *where* $s = ex_site(T_i)$. Upon receiving message $(T_i, ws(T_i), rs(T_i))$, register T_i with $CS_s(C)$ without using timestamp. (Being a cautious scheduler, $CS_s(C)$ always accepts T_i.)

A4. *Registration of a global transaction T_i sent from site l.* Let the received message be $(T_i, ws(T_i) \cap \overline{D}_s, rs(T_i) \cap \overline{D}_s)$. Register T_i with $CS_s(C)$ with timestamp $t_l(T_i) = (cl_l, id(l))$ (as noted in Section 8.1, this registration will always be successful), and return message $(T_i, t_l(T_i))$ to l.

A5. *Request for executing a physical operation of T_i sent from site l.* If timestamp $t_{max}(T_i)$ accompanies it, change the timestamp of T_i in $CS_s(C)$ to $t_{max}(T_i)$. (As Theorem 8.1 will show, this change does not create any cycle in $atio^*_s(\cdot)$). Let the clock part of $t_{max}(T_i)$ be cl and the current clock value of site s be cl_s. Advance cl_s to cl if $cl > cl_s$. Otherwise (i.e., T_i is a local transaction or $t_{max}(T_i)$ is already known to $CS_s(C)$), do not change its timestamp. Execute the physical operation and send the result back to site l. □

Note 1. In the above description, site l may be the same as site s. In this case, the message exchanges between s and l are not necessary, but the rest of the procedures must be carried out as stated.

Note 2. If the operation in question is a physical write operation, the value to be written must accompany the request for its execution, though this is not explicitly stated in the above description. Similarly, if the operation is a physical read operation, the result read from the copy must accompany the reply message.

Note 3. The cautious scheduler $CS_s(C)$ at each site s is the same as the cautious scheduler in [IKK88, IKK89] designed for centralized database systems (see Section 4), except that some constraint arcs due to timestamps are added to the ATIO graph. This change may sometimes make the registration of a transaction T_i not possible, because the addition of a node T_i with the constraint arcs due to timestamps may create cycles in the ATIO graph. This is why the timestamp is reset to a greater value in A1(3) and A5. This procedure will be justified in the next section.

Note 4. As local transactions have no timestamps, they have a higher priority than global transactions in the following sense. A local transaction is immediately registered at its execution site, without any communication or other processing overhead required to determine its timestamp; thus steps A1(3) and A4 can be skipped for local transactions. Also, in cautious scheduling by $CS_s(C)$, local transactions are less constrained since they have no constraint arcs due to timestamps and hence have higher probability of being immediately granted when they request the execution of their steps. In particular, if all transactions are local, no timestamp is introduced and all sites work as independent local database systems.

Note 5. Some unnecessary delay may be encountered due to the use of tentative timestamps. For example, let T_i, T_j, and T_k be transactions which access x_s at site s with timestamps $t_{max}(T_i) < t(T_j) < t_{max}(T_k)$, where $t(T_j)$ is a tentative timestamp. Suppose that $w_i[x_s]$ has already been carried out. If a request for $r_k[x_s]$ arrives at site s, s must delay this request until $t_{max}(T_j)$ becomes known, and it may

later turn out that $t_{max}(T_j) > t_{max}(T_k)$, in which case the waiting was really unnecessary. To reduce such a waiting period, it may be advantageous to broadcast $t_{max}(T_j)$ as soon as it is determined, without waiting until the first request by T_j is sent.

8. Correctness of Algorithm A

We first need to prove a few properties of Algorithm A in order to show that Algorithm A outputs only serializable schedules without ending up deadlocked or necessitating rollbacks. We do this in three subsections; the first subsection shows that the mechanism for handling timestamps guarantees the proper functioning of cautious schedulers $CS_s(C)$ at each site s, the second subsection shows that Algorithm A outputs only serializable schedules, and the final subsection proves that Algorithm A does not lead to any deadlock.

8.1. Timestamp assignment

As stated in Section 5(2), no two global transactions are assigned the same timestamp. It is, however, possible that one global transaction receives more than one timestamp (one at a time) in the registration phase (see steps A4 and A5 of Algorithm A). In the end, however, transaction T_i receives its final timestamp $t_{max}(T_i)$ determined by step A1(3), which is unique to this transaction. No timestamps are assigned to local transactions.

It is necessary to show that adding a node T_i to acyclic $atio_s^*(\cdot)$ together with the associated constraint arcs in Step A4, in response to a registration request from T_i, does not create a cycle. It is easy to see that node T_i has the largest timestamp in $atio_s^*(\cdot)$, implying that there is no constraint arc due to timestamps, which goes out of node T_i. Also, in the registration phase, all physical operations of T_i that will be executed at site s are placed in *PEND*, because execution requests for them have not arrived yet. This means that none of the read-from arcs, *ww*-constraint arcs or *wr*-constraint arcs (see Section 4), goes out of node T_i. Therefore, T_i cannot be involved in any cycle in $atio^*(\cdot)$, and hence $atio^*(\cdot)$ remains acyclic.

The timestamp given to a transaction T_i in $atio^*(\cdot)$ is sometimes advanced to a larger value (see steps A1(3) and A5). The next theorem shows that this change does not prevent $CS_s(WW)$ or $MCSs(MWW)$ from functioning properly.

Theorem 8.1 Consider a cautious scheduler $CS_s(WW)$ or $MCS(MWW)$. If $atio^*(\cdot)$ is acyclic, the ATIO graph obtained from it by advancing the timestamp of a node T_i is also acyclic, provided that the physical operations of T_i to be executed at site s are currently all in *PEND* (as is the case in steps A1(3) and A5).

Proof. Assume that $atio^*(\cdot)$ after advancing timestamp t_i of T_i to $t_i'(> t_i)$ has a cycle involving T_i of the form, $...\rightarrow T_g \rightarrow T_i \rightarrow T_h...$ Observe now that no new constraint arc out of T_i is created by the above change of timestamp. Furthermore, a new constraint arc (T_j, T_i) entering T_i can be created only if it is a constraint arc due to timestamps, which satisfy $t_i < t_j < t_i'$, where t_j is the timestamp of T_j, or it is an exclusion arc resulting from the new constraint arcs due to timestamps. Therefore, we can assume without loss of generality that (T_g, T_i) is a constraint arc due to timestamps, which is newly created. Next, since all physical operations of T_i are in *PEND*, none of the reads-from arcs and *ww*-, *wr*-constraint arcs goes out of T_i. Similarly, no exclusion arc goes out of T_i, since T_i has no incoming reads-from arc. Therefore, (T_i, T_h) is also a constraint arc due to timestamps. However, if both (T_g, T_i) and (T_i, T_h) are constraint arcs due to timestamps, so must (T_g, T_h) also be. This implies that there was a cycle not involving T_i in the original $atio_s^*(\cdot)$, a contradiction. □

8.2. Serializability

Let $<\alpha, \overline{I}>$ be a physical schedule output by Algorithm A, where α is an RD log and $\overline{I}$ represents the version selection made by the cautious schedulers at all sites in the system. We shall show that this $<\alpha, \overline{I}>$ is 1SR, i.e., it realizes a certain serializable schedule $<\sigma, I>$. By Theorem 3.2, this can be done by showing that $tio(\alpha, \overline{I})$ has a DITS.

Theorem 8.2 If $<\alpha, \overline{I}>$ is output by Algorithm A, then $tio(\alpha, \overline{I})$ has a DITS.

Proof. Let $tio_s(\sigma_s, \overline{I}_s)$ denote the TIO graph at site s obtained after executing all the requested operations (i.e., the final ATIO graph at site s). In σ_s, the initial step $w_0[\overline{D}_s]$[5] (a physical step of T_0 at site s) writes the initial values of all copies at site s and the final step $r_n[\overline{D}_s]$ (a physical step of T_n at site s) reads their final values. Cautious scheduling performed by $CS_s(C)$ or $MCS_s(MC)$ guarantees that each $tio_s(\sigma_s, \overline{I}_s)$ has a DITS that orders T_0 first and T_n last.

We now construct $tio(\alpha, \overline{I})$ by superimposing all $tio_s(\sigma_s, \overline{I}_s)$ as follows. For each i $(0 \le i \le n)$, coalesce all nodes T_i into a single node T_i. T_0 and T_n appear in each *tio* graph, and a global transaction T_i appears in the *tio* graphs of its relevant sites. All dummy nodes are simply added to the graph being constructed. All arcs are also simply copied, except that duplicate constraint arcs are removed. Add a constraint arc (T_i, T_j) due to timestamps if global transaction T_i and T_j are not simultaneously in any *tio* graph. It is easy to see that the resulting graph is $tio(\alpha, \overline{I})$.

On the set of nodes of $tio(\alpha, \overline{I})$ thus obtained, find a topological sort $\ll$ which orders T_0 first and T_n last, and is consistent with a DITS on $tio_s(t_s, \overline{I}_s)$ at every site s. Such a total order exists because each

[5] To be precise, only those copies that are accessible within the current view are initialized by T_0.

$tio_s(\sigma_s, \overline{I}_s)$ has a DITS which is consistent with the timestamp order (as a result of the constraint arcs due to timestamps) and which orders T_0 first and T_n last.

We now show that this $\ll$ is a DITS for $tio(\alpha, \overline{I})$. Property (i) of a DITS (see Section 3) is an obvious consequence of the above construction. Property (ii') can be shown as follows. Consider two arcs $(T_g, T_h):x_s$ and $(T_i, T_j):x_{s'}$ in $tio(\alpha, \overline{I})$, where $T_g \neq T_i$, and x_s and $x_{s'}$ are copies of a data item X at sites s and s', respectively. This means that $tio_s(\sigma_s, \overline{I}_s)$ (resp. $tio_{s'}(\sigma_{s'}, \overline{I}_{s'})$) contains an arc $(T_g, T_h):x_s$ (resp. $(T_i, T_j):x_{s'}$). If $s = s'$, these obviously satisfy property (ii'), since both are in one $tio_s(\sigma_s, \overline{I}_s)$ and its DITS satisfies (ii'). If $s \neq s'$, on the other hand, recall that T_g and T_i are both global transactions writing a data item X whose copies are located at sites s and s' (and possibly others). Therefore, both $tio_s(\sigma_s, \overline{I}_s)$ and $tio_{s'}(\sigma_{s'}, \overline{I}_{s'})$ contain nodes T_g and T_i, by the quorum selection rule in Section 2. As T_g and T_i are ordered by time stamps, assume without loss of generality that there is a path from T_g to T_i in both TIO graphs. This means that only the order T_g, T_h, T_i is possible in $tio_s(\sigma_s, \overline{I}_s)$, and only the order T_g, T_i, T_j is possible in $tio_{s'}(\sigma_{s'}, \overline{I}_{s'})$ among T_g, T_h, T_i and T_j. Therefore, the DITS $\ll$ for $tio(\alpha, \overline{I})$ must satisfy $T_g \ll T_h \ll T_i \ll T_j$. This proves property (ii'). □

8.3. Deadlock-freedom

As a characterizing feature of cautious scheduling, cautious scheduler $CS_s(C)$ at each site s never deadlocks because of the operations executed at site s. However, there might occur a global deadlock that involves more than one site. Such a possibility is illustrated in Fig. 4, in which a physical operation p_{i1} waits at site a for a physical operation p_{j2} to be executed, because p_{i1} is currently placed in the set DEL of $CS_a(C)$ and cannot be executed unless p_{j2} (the request for which has not arrived yet) is completed, p_{i2} waits for p_{i1} to be executed because p_{i1} precedes p_{i2} in transaction T_i (recall that a transaction can request the execution of a step only if all the preceding steps have been executed), and so forth. As these waits-for arcs form a cycle, no operation in the cycle can be executed, i.e., a deadlock exists.

In general, this situation is quite possible. However, for cautious schedulers $CS(WW)$ and $MCS(MWW)$ (and for some other cautious schedules of interest [IKK88, IKK89]), we can prove the next lemma, based on which Theorem 8.3 shows that no deadlock as illustrated above can occur.

Lemma 8.1 Consider a cautious scheduler $CS_s(WW)$ or $MSC_s(MWW)$, and let T_i and T_j be global transactions with timestamps $t_i < t_j$, respectively. Then, under Algorithm A, T_i does not wait for T_j. In other words, the execution of a physical operation of T_i does not depend on the status of T_j, i.e., how many of the physical operations of T_j have been executed.

Proof. Consider $atio_s(<P, \bar{I}>, q, PEND)$ at certain time instant. Take a DITS $\ll$ for this graph. By the constraint arcs due to timestamps, $T_i \ll T_j$ holds for any pair of global transactions T_i and T_j with $t_i < t_j$. Now let β be the sequence of all physical operations in $PEND \cup \{q\}$ (those operations are called **pending**) arranged in the order of their issuing transactions in DITS $\ll$ (i.e., all the pending physical operations of the first transaction, then all the pending physical operations of the second transaction, and so forth). Then, for classes $C = WW$ and MWW, it can be shown that $P\beta \in C$ holds [IKM87, IKK88, IKK89]. This means that all physical operations in $PEND \cup \{q\}$ can be executed in the order of β, and hence the execution of a pending physical operation of T_i can be granted without regard to the status of T_j, because all the pending physical operations of T_j come after those of T_i in β. □

Theorem 8.3 If cautious scheduler $CS(WW)$ or $MCS(MWW)$ is used at each site, Algorithm A does not cause any global deadlock.

Proof. In a deadlock as illustrated in Fig. 4, $t_i < t_j$ must hold by Lemma 8.1 if a physical operation of T_j waits for a physical operation of T_i to be executed at site s, where t_i (resp. t_j) is the timestamp of T_i (resp. T_j). A cycle shown in Fig. 4 therefore implies $t_i < t_i$, but this is not possible. □

9. Algorithm B

We now present our second algorithm, Algorithm B, for distributed cautious scheduling. This one is simpler than Algorithm A, but can be used only when the following conditions are met.

(A) $D_s \cap D_{s'} = \emptyset$ for any pair of sites $s \neq s'$, i.e., there are no duplicate copies of a data item and thus each read quorum and write quorum is a singleton.

(B) Each site s uses a multiversion WW cautious scheduler, $MCS_s(MWW)$.

(C) Each transaction T_i has its execution site $ex_site(T_i)$ (which may be different from $home(T_i)$ where T_i enters the system). T_i can issue any read operation $R_i[X]$, but can issue a write operation $W_i[X]$ only if the copy of X is located at $ex_site(T_i)$.

These are restrictive. Conditions (A) and (B) can be taken into account in the design stage of the system. Condition (C) restricts the transactions that users can submit. It is argued in [CKI88, KoG87], which discuss "fragmented databases," that this condition is physically met in many important applications in the real world.

Basically, Algorithm B is a simplified version of Algorithm A in that the adjustment of the timestamps in the registration phase of Algorithm A (i.e., steps A1, A4 and A5 as illustrated in Fig. 3(a) and (b)) is not necessary. This significantly reduces the number of messages exchanged among sites, thereby speeding up transaction processing. In Algorithm B, the cautious scheduler $CS_s(MWW)$ at each site s

works almost independently of the other sites, except that the timestamps of global transactions cause some interaction among cautious schedulers.

Algorithm B for the transaction manager TM_s at site s consists of the following steps.

B1. *Arrival of a transaction T_i.* Site s becomes *home* (T_i). Based on the sets $WS(T_i)$ and $RS(T_i)$ predeclared by T_i, determine *ex_site* (T_i) and send T_i to *ex_site* (T_i) for registration. (If T_i has a write operation, *ex_site* (T_i) is uniquely determined by condition (C); otherwise a site is chosen by some appropriate rule from among those sites where read operations of T_i take place.) If T_i is a global transaction (i.e., has a read operation at a site different from *ex_site* (T_i)), request a timestamp for T_i, $t(T_i)$, from *ex_site* (T_i). (T_i can proceed to B2 only after B1 is completed.)

B2. *Submission of a step from T_i with home $(T_i) = s$.* Decompose the step into physical operations and send these operations to the sites where the copies of data items they access are located; if the destination is not *ex_site* (T_i) and if this is the first request of T_i to the site, send $t(T_i)$ as well. Deliver the results received from these sites to T_i.

B3. *Registration request for T_i sent from site l = home (T_i), where s = ex_site (T_i).* If T_i is a local transaction, register T_i without a timestamp immediately with $CS_s(MWW)$. If T_i is a global transaction, on the other hand, define the timestamp of T_i by

$$t(T_i) = (cl_s, id(s)),$$

where cl_s is the current clock value of site s, and then register T_i (together with $t(T_i)$) with $CS_s(MWW)$. (The timestamp given to T_i is permanent.) Return $t(T_i)$ to site l.

B4. *Request for the execution of a physical operation of T_i, sent from site l = home (T_i).* If the timestamp $t(T_i)$ accompanies the request, then this is the first request to site s from T_i; (i) register T_i (with $t(T_i)$) with $CS_s(MWW)$, and (ii) if the clock part cl of $t(T_i)$ is larger than the current clock value cl_s of site s, advance cl_s to cl. Input the received physical operation to $CS_s(MWW)$. Return the result to site l. □

Fig. 5 depicts typical rounds of communication for the registration of T_i and for executing a step issued by T_i. If the step is a write step, condition (C) above implies that the execution site is always *ex_site* (T_i). It will often be the case that *home* (T_i) = *ex_site* (T_i); in this case some message exchanges described above are unnecessary. The notes given at the end of Section 7 also apply to Algorithm B.

10. Properties of $MCS_s(MWW)$ and the Correctness of Algorithm

Before proving the correctness of Algorithm B, we review some properties of the cautious schedulers, which play an essential role here.

During the execution of T_i, it may become necessary for T_i to change its read and write sets to $RS'(T_i)$ and $WS'(T_i)$, respectively, from the declared $RS(T_i)$ and $WS(T_i)$. The blocking of a cautious scheduler caused by $WS'(T_i) \subset WS(T_i)$ or $RS'(T_i) \subset RS(T_i)$ is called the **cancellation anomaly**. It is shown in [IKK88, IKK89] that most of the cautious schedulers, including $CS_s(WW)$ and $MCS_s(MWW)$, do not exhibit cancellation anomaly. Similarly, the malfunction caused by $RS(T_i) \subset RS'(T_i)$ (i.e., some additional data items are requested for read operations) is called the **(read) augmentation anomaly**. For the particular cautious scheduler $MCS_s(MWW)$, useed in Algorithm B, the following theorem is known.

Theorem 10.1 [IKK89] The cautious scheduler $MCS_s(MWW)$ does not exhibit augmentation anomaly. □

It should be noted here that the augmentation of the write set, i.e., $WS(T_i) \subset WS'(T_i)$, cannot be allowed in $MCS_s(MWW)$. Now to prove the correctness of Algorithm B, we begin with the next theorem.

Theorem 10.2. The registration of T_i with a cautious scheduler $MCS_s(MWW)$ in Algorithm B always succeeds.

Proof. Three cases are possible :

T_i is a local transaction at $ex_site(T_i)$ (B3). In this case, T_i carries no timestamp, and therefore, any cautious scheduler, in particular $MCS_s(MWW)$, will accept it.

T_i is a global transaction and $s = ex_site(T_i)$ (B3). The mechanism of adjusting the clocks (see B4(ii)) guarantees that the timestamp $t(T_i)$ issued at s is the largest among those already registered with $MCS_s(MWW)$. Therefore, T_i is accepted by $MCS_s(MWW)$ as discussed in the second paragraph of Section 8.1.

T_i is a global transaction with $s \neq ex_site(T_i)$ (B4(i)). In this case, timestamp $t(T_i)$ accompanies the read request from T_i, but $t(T_i)$ may not be the largest among those already registered with $MCS_s(MWW)$. However, T_i is accepted into $MCS_s(MWW)$ because T_i has only physical *read* operations (i.e., no write operations) to be executed at site s (by condition (C)), and $MCS_s(MWW)$ has no augmentation anomaly (Theorem 10.1). (Consider that T_i was already (fictitiously) registered at time $t(T_i)$ with the empty read and write sets, and, at the this moment, it has augmented its read set.) □

Once $MCS_s(MWW)$ accepts a transaction T_i, it executes each physical operation requested by T_i in finite time, since, in Algorithm B, the timestamp given to T_i never changes. In view of this, we only need the following two theorems in order to show that Algorithm B is correct.

Theorem 10.3 Redefine the graph $tio(\alpha, \overline{I})$, defined in the beginning of Section 8.2, now for the distributed database system operated by Algorithm B. Then $tio(\alpha, \overline{I})$ has a DITS. □

The proof of Theorem 8.2 carries over here with the obvious modifications. Theorem 3.2 then implies that the physical schedule $<\alpha, \overline{I}>$ eventually output by Algorithm B is 1SR.

Theorem 10.4 Algorithm B never causes global deadlocks. □

The proofs of Lemma 8.1 and Theorem 8.3 carry over.

11. Algorithm C

In implementing Algorithm B, we may sometimes want to give a higher priority to local transactions than global transactions in the following sense: a local transaction never needs to wait for the execution of steps requested by a global transaction at a remote site. This is important since in some applications most transactions are local, and the execution of a step of a remote global transaction may require a significant amount of time just for communication. The fragmented distributed database system [CKI88, KoG87] also aims to achieve a similar objective.

To give a higher priority to local transactions, we further require in this section (in addition to the conditions (A)(B)(C) in Section 9) that every global transaction consist of at most one read step followed by at most one write step:

$$T_i = R_i[S]W_i[S]$$

The physical read operations corresponding to $R_i[S]$ are classified into **local** read operations that are executed at its execution site, and **remote** read operations that are executed at sites different from its execution site. In the modification to be discussed below, we execute all the remote read operations of T_i before its local read operations. When a global transaction T_i with $s = ex_site(T_i)$, as well as all the global transactions T_j with $s = ex_site(T_j)$ and $t(T_j) < t(T_i)$, has completed all the remote read operations, then T_i is said to be **localized**. Once it is localized, a global transaction T_i is treated as if it were a local transaction, because the rest of its operations are all executed at $ex_site(T_i)$.

We now design Algorithm C by modifying Algorithm B by changing rule B2 to B2' and adding a new rule B5' given below. B2' and B5' together state that the decomposition of a step submitted by T_i into physical operations and their distribution to the corresponding sites are now carried out by its execution site $ex_site(T_i)$ (instead of $home(T_i)$), and that some additional control is introduced in order to

handle global transactions on the basis of whether they are localized or not.

B2'. *Submission of a step by T_i.* Send the step to $ex_site(T_i)$.

B5'. *Arrival of a step of T_i from site l = home (T_i), where s = ex_site (T_i).*

(1) If T_i is a local transaction, decompose the step into physical operations and give them to $MCS_s(MWW)$ for execution. Return the results to site l.

(2) If T_i is a global transaction and the step is its first step, i.e., $R_i[S]$, then decompose this step into physical read operations. Send all the remote read operations, together with the timestamp $t(T_i)$, to the appropriate sites for execution. Wait until T_i is localized, i.e., T_i has completed all the remote read operations, and all other global transactions T_j with $s = ex_site(T_j)$ and $t(T_j) < t(T_i)$ are localized. Give the remaining local operations of $R_i[S]$ to $MCS_s(MWW)$ for execution. Return the results of $R_i[S]$ to site l.

(3) If T_i is a global transaction and the step is its second step, i.e., $W_i[S]$. Decompose $W_i[S]$ into physical operations and give them to $MCS_s(MWW)$ for execution. Return the results to site l. □

It is noted that step B5'(2) is necessary even if $RS(T_i)$ is empty (in this case, B5'(2) says that T_i simply waits until all other global transactions T_j with $s = ex_site(T_j)$ and $t(T_j) < t(T_i)$ are localized).

The proof that this new algorithm always outputs a serializable schedule and that no deadlock can occur is similar to that for Algorithm B. The next theorem shows that, in the new algorithm, the local and localized transactions physically enjoy a higher priority than other global transactions.

Theorem 11.1 Under Algorithm C, no operation of a local transaction or a localized global transaction waits at $MCS_s(MWW)$ for the execution of a remote read operation of a global transaction.

Proof. Assume that an operation q of a local transaction or a localized global transaction was not immediately granted by $MCS_s(MWW)$, i.e., $G^* = atio_s{}^*(\langle P, \overline{I}\rangle, q, PEND)$ contained a cycle. Let $\mathbf{T}_s'$ denote the set of global transactions which have been registered at site s but not localized yet. The theorem is proved if the above cycle in G^* does not contain any node $T_i \in \mathbf{T}_s'$, since this means that q must be delayed even if no $T_i \in \mathbf{T}_s'$ is there.

Now consider a node $T_i \in \mathbf{T}_s'$ in G^*. By B5'(2), all the operations of T_i to be executed at site s belong to $PEND$. Therefore, no reads-from arc enters node T_i, and none of the reads-from and *ww*-constraint arcs leaves such node T_i. Similarly, since a global transaction registered at site s is localized only when all global transactions registered at s with smaller timestamps are localized, no constraint arc due to timestamps is drawn from T_i to $T_j \notin \mathbf{T}_s'$. This means that there is no path from T_i to any node $T_j \notin \mathbf{T}_s'$ in $atio_s(\langle P, \overline{I}\rangle, q, PEND)$ (i.e., before adding exclusion arcs). By definition of exclusion arcs

(see Section 4), therefore, there is no exclusion arc from T_i to any node $T_j \notin \mathbf{T}_s'$ in G^*.

We thus conclude that the assumed cycle in G^* involves no $T_i \in \mathbf{T}_s'$. □

12. Discussions and Conclusions

Three distributed concurrency control algorithms, Algorithms A, B and C, have been presented in this paper. Algorithm A is the most general in the sense that no restriction is imposed on transactions, and almost any cautious scheduler can be used at each site (though *CS*(*WW*) and *MCS*(*MWW*) are used in this paper for illustration purposes). Algorithms B and C can be used only in a more restricted setting, but they are much simpler than Algorithm A and incur less communication overhead for concurrency control.

In all the above three algorithms, as mentioned in Note 4 (Section 7), each site becomes an independent local database system if no global transaction is executed in the system. This suggests that, in many applications in which a large majority of the transactions are local, the overall performance of the distributed system would not be much hampered by the existence of few global transactions. Further experimental or simulational studies are, however, necessary to confirm this claim.

There is another point worth noting. In cautious scheduling, each transaction is assumed to faithfully predeclare its write and read sets. In reality, this may be difficult to enforce. As briefly mentioned in Section 10, cautious schedulers as used in this paper do not exhibit "cancellation anomaly" individually. In other words, it is harmless for the transactions to cancel some of the predeclared operations. It can be shown that none of the three algorithms exhibits cancellation anomaly as a whole if each site employs a cautious scheduler that is free of cancellation anomaly.

Similar property holds for the "read augmentation anomaly". In Algorithms B and C, the read set can be augmented with new data items not predeclared. Note, however, that this is not the case for Algorithm A (since all relevant sites must participate in the registration phase in which the timestamp is determined). In general, no algorithm is free from write augmentation anomaly. References [IbK87, IbK88] discuss how to deal with write augmentation for single site cautious schedulers. The approach used there can be generalized to the distributed case, which is of interest here. In this case, however, rollbacks of some transactions become necessary; but global deadlocks (i.e., those involving more than one site) can still be avoided.

In situations where a transaction may change its readset or writeset, it is necessary for each transaction to send the "exit message" when all its steps are completed. Its home site then sends the exit message to all the relevant sites, so that the cautious schedules there know its exit from the system. Although

this point is not included in the algorithm descriptions in this paper, it is straightforward to implement it.

References

[BHG87] P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, MA, 1987.

[CaB80] M. A. Casanova and P. A. Bernstein, General purpose schedulers for database systems, *Acta Informatica 14*, (1980), 195-220.

[ChK85] D. Cheung and T. Kameda, Site-optimal termination protocols for a distributed database under network partitioning, *Proc. 4th ACM Symp. Distributed Computing*, Minaki, Ont., Aug. 1985, 111-121.

[CKI88] D. Cheung, T. Kameda, T. Ibaraki and A. Fu, A concurrency control scheme for fragmented database systems, CSS/LCCR Tech. Rept. TR88-19, Simon Fraser University, 1988.

[ChR86] F. Chin and K. V. S. Ramarao, Optimal termination protocols for network partitioning, *SIAM Journal of Computing 15*, 1 (Feb 1986), 131-144.

[ESC85] A. El-Abbadi, D. Skeen and F. Cristian, An efficient, fault-tolerant protocol for replicated data management, *Proc. 4th ACM Symp. Principles of Database Systems*, Portland, Ore., March 1985, 215-229.

[EIT86] A. El-Abbadi and S. Toueg, Availability in partitioned replicated databases, *Proc. 5th ACM Symp. Principles of Database Systems*, Cambridge, MA, March 1986, 240-251.

[EIT89] A. El-Abbadi and S. Toueg, Maintaining availability in partitioned replicated databases, *ACM Trans. Database Systems 14*, (1989), To appear.

[FuK89] A. Fu and T. Kameda, A family of virtual partition protocols for replicated database systems, School of Computing Science, Simon Fraser University, Manuscript, 1989.

[Gif79] D. K. Gifford, Weighted voting for replicated data, *Proc. 7th ACM Symp. OS Principles*, Pacific Grove, CA, Dec. 1979, 150-162.

[GoL87] K. J. Goldman and N. A. Lynch, Quorum consensus in nested transaction systems, *Proc. 6th ACM Symp. Principles of Distributed Computing*, Vancouver, B.C., Aug. 1987, 27-41.

[Her87] M. Herlihy, Dynamic quorum adjustment for partitioned data, *ACM Trans. Database Systems 12*, 2 (Sept. 1987), 170-194.

[IKM87] T. Ibaraki, T. Kameda and T. Minoura, Serializability with constraints, *ACM Trans. Database Systems 12*, 3 (Sept. 1987), 429-452.

[IbK87] T. Ibaraki and T. Kameda, Cautious schedulers under imcomplete predeclarations (in Japanese), Proc. 3rd Data Engineering Seminar, IEICE of Japan, Kyoto, Nov. 1987.

[IKK88] T. Ibaraki, T. Kameda and N. Katoh, Cautious transaction schedulers for database concurrency control, *IEEE Trans. Software Eng. 14*, 7 (July 1988), 997-1009.

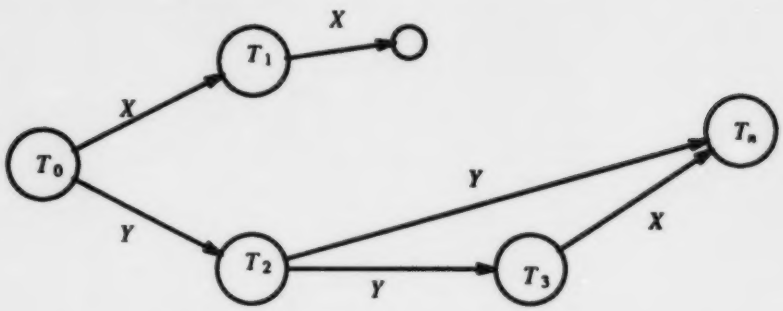
[IbK88] T. Ibaraki and T. Kameda, Dealing with inaccurate predeclarations in cautious scheduling (in Japanese), Database System Group of the Information Processing Society of Japan, 1988.

[IKK89] T. Ibaraki, T. Kameda and N. Katoh, Cautious transaction schedulers for multiversion database systems, LCCR Tech. Rep. 86-12, Simon Fraser University, To appear in IEEE Trans. Software Eng., 1989.

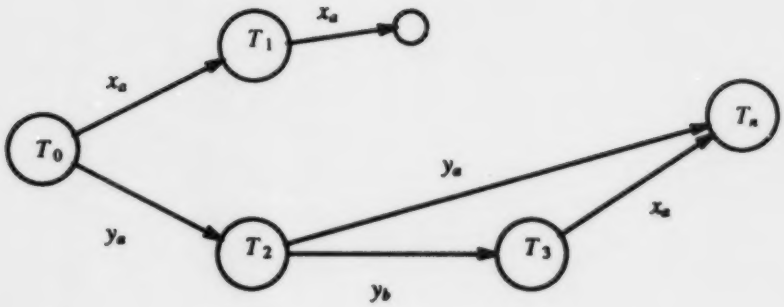
[KoG87] B. Kogan and H. Garcia-Molina, Update propagation in Bakunin data networks, *Proc. 6th ACM Symp. Principles of Distributed Computing*, Vancouver, B.C., Aug. 1987, 13-26.

[NTI89] S. Nishio, S. Taniguchi and T. Ibaraki, On the efficiency of cautious schedulers, Tech. Rept. #89002, Dept. Applied Math & Physics, Kyoto University, 1989.

[Pap86] C. H. Papadimitriou, *The Theory of Concurrency Control*, Computer Science Press, Rockville, MD, 1986.



(a) $TIO(\sigma, I)$



(b) $tio(\alpha, \overline{I})$

Figure 1. TIO graphs $TIO(\sigma, I)$ and $tio(\alpha, \overline{I})$ of Example 3.1.
(The small nodes represent dummy nodes.)

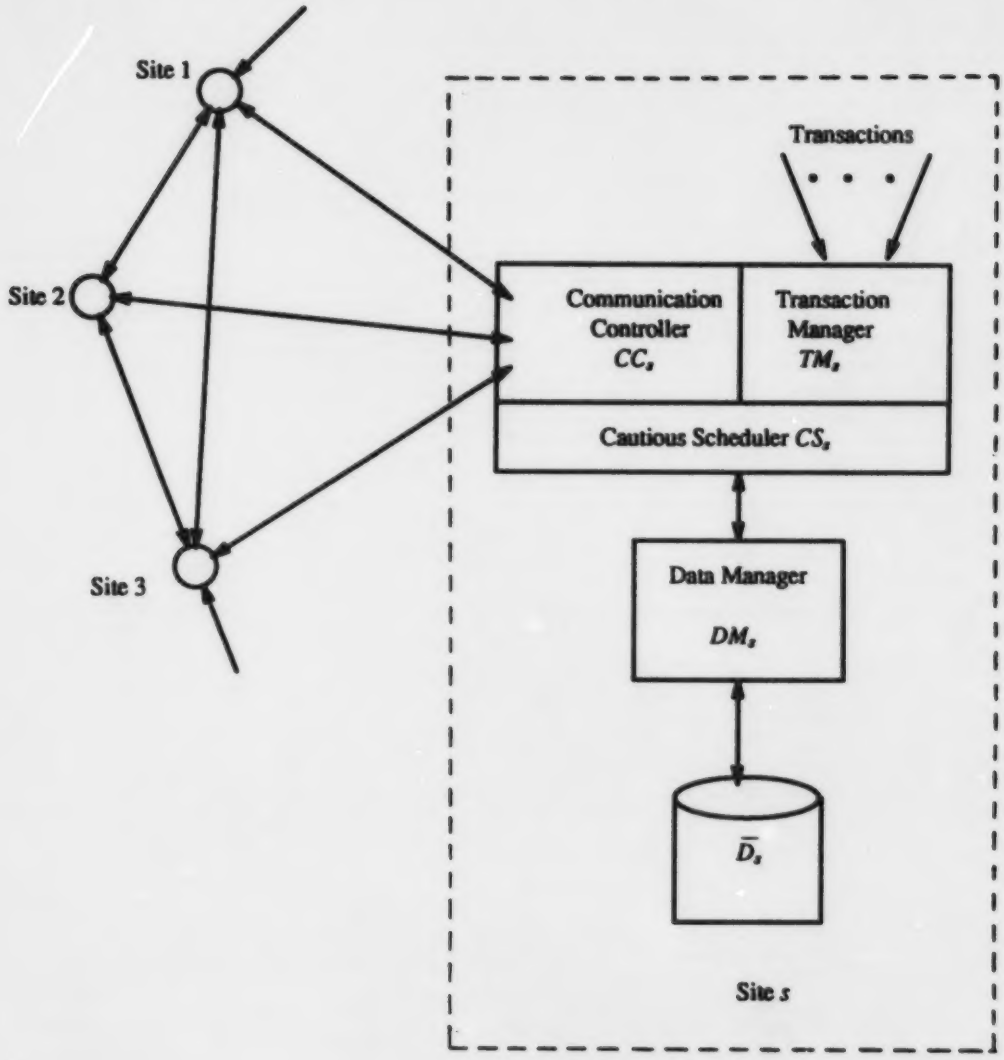
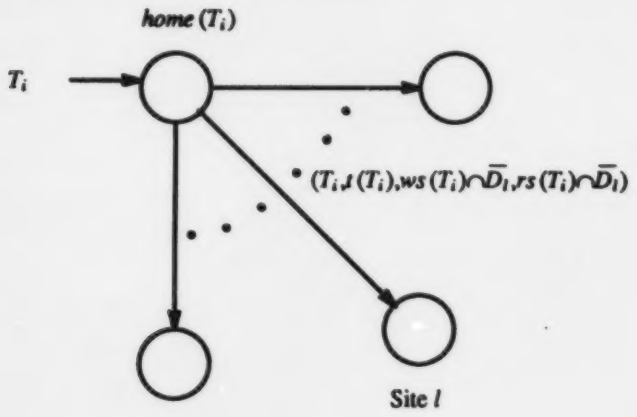
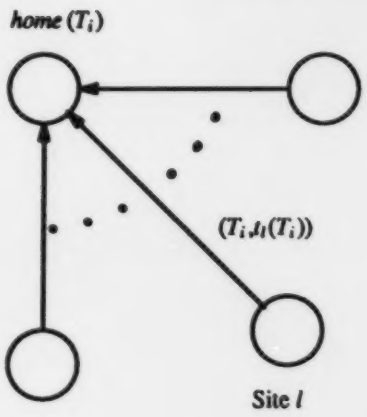


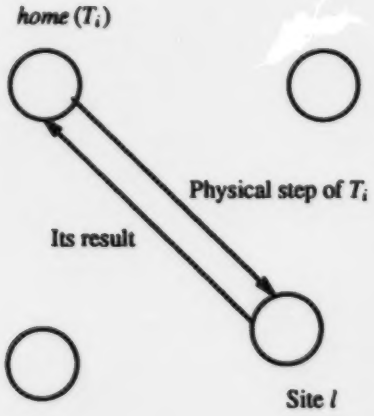
Figure 2. Distributed database management system.



(a) Registration request.

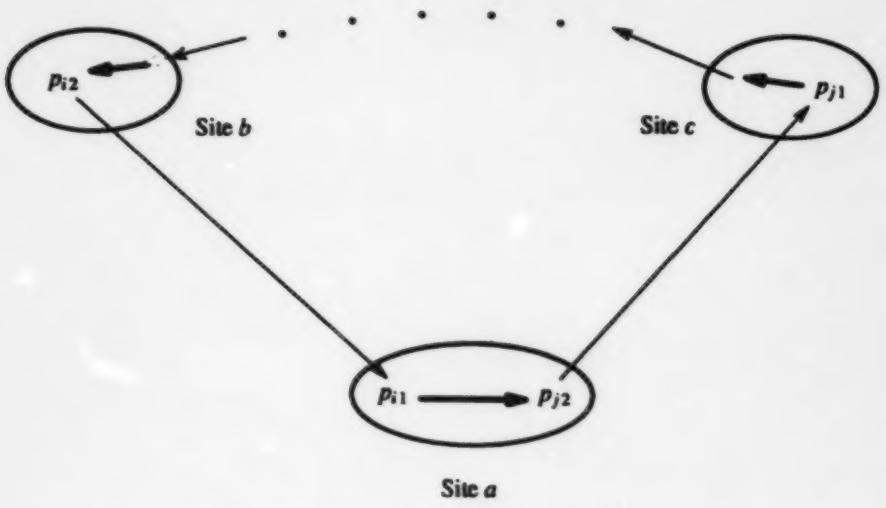


(b) Registration completion.



(c) Processing a step.

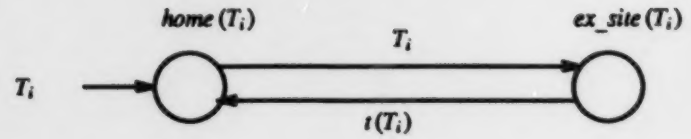
Figure 3. Registration and processing of a global transaction by Algorithm A.



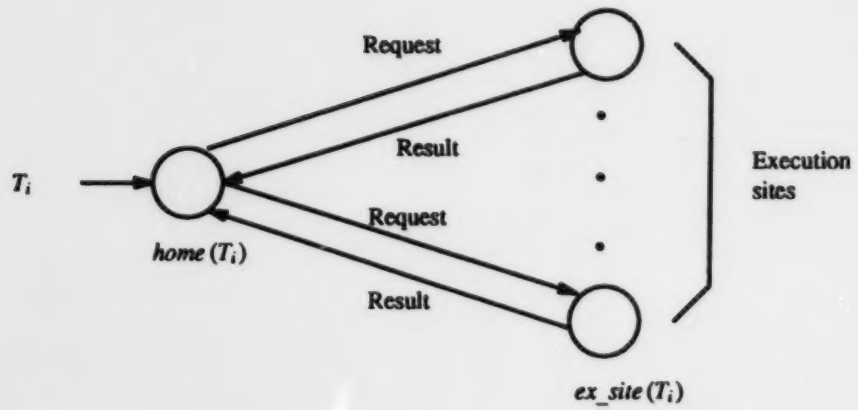
Waits-for arc within the same transaction

Waits-for arc between different transactions

Figure 4. A potential deadlock involving more than one site.



(a) Registration of a global transaction T_i.



(b) Execution of a step of T_i

Figure 5. Message exchanges for registration and execution in Algorithm B.

Centre for Systems Science

• Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.

• CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.

• CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.